**Universidad Nacional de Colombia - Sede Bogotá**
**Facultad de Ingeniería**
**Departamento de Ingeniería de Sistemas e Industrial**
**Curso: Ingeniería de Software**
**Informe del linter**

---

**Grupo:** Los Inadaptados 🪵

**Estudiantes:**

- Camilo Lagos Malaver
- John Freddy Morenazo Alejo
- Juan Felipe Marin Moreno
- Sergio Nicolas Escobar

---

## Pruebas unitarias 🧩

Para la realización de los *unit tests*, se utilizó la librería XUnit ⚠️ para C#. A continuación se describen brevemente las pruebas que se diseñaron en cada caso; no obstante, antes es conveniente entender algunas líneas que aparecen con frecuencia:

🧪 [Fact] (prueba estándar)

🔬 [Theory] (prueba parametrizada)

📊 [InlineData] (datos de entrada)

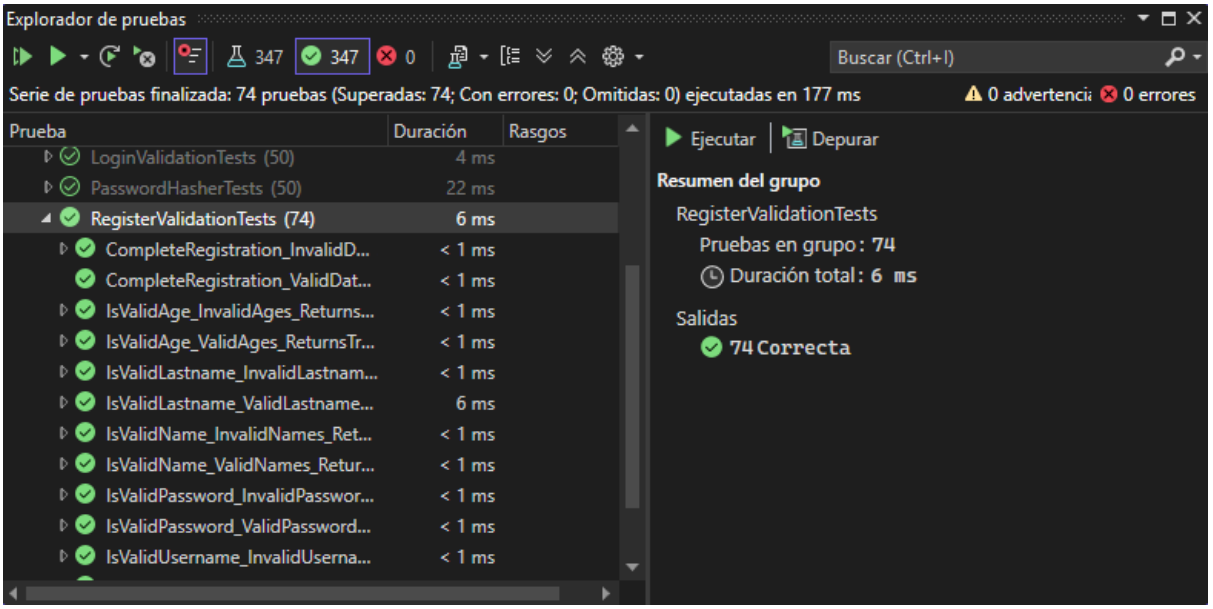
Las pruebas se dividen en dos tipos: pruebas individuales ([Fact]) para casos específicos y pruebas parametrizadas ([Theory]) que evalúan múltiples valores, incluyendo entradas válidas e inválidas con explicaciones claras sobre los errores. Además, emplea ITestOutputHelper para registrar resultados detallados durante la ejecución, lo que facilita la identificación de fallos.

Ya con esto, se documentan ahora las pruebas realizadas.

1. **Registro (RegisterValidationTests.cs) ✔️**

   Este código es un conjunto de pruebas unitarias diseñadas para validar los campos de un formulario de registro, como nombre de usuario, contraseña, nombre, apellido y edad. El objetivo principal es asegurar que las reglas de validación funcionen correctamente, como rechazar usernames con espacios o contraseñas sin números. También incluye pruebas de escenarios completos, donde se verifica que todos los

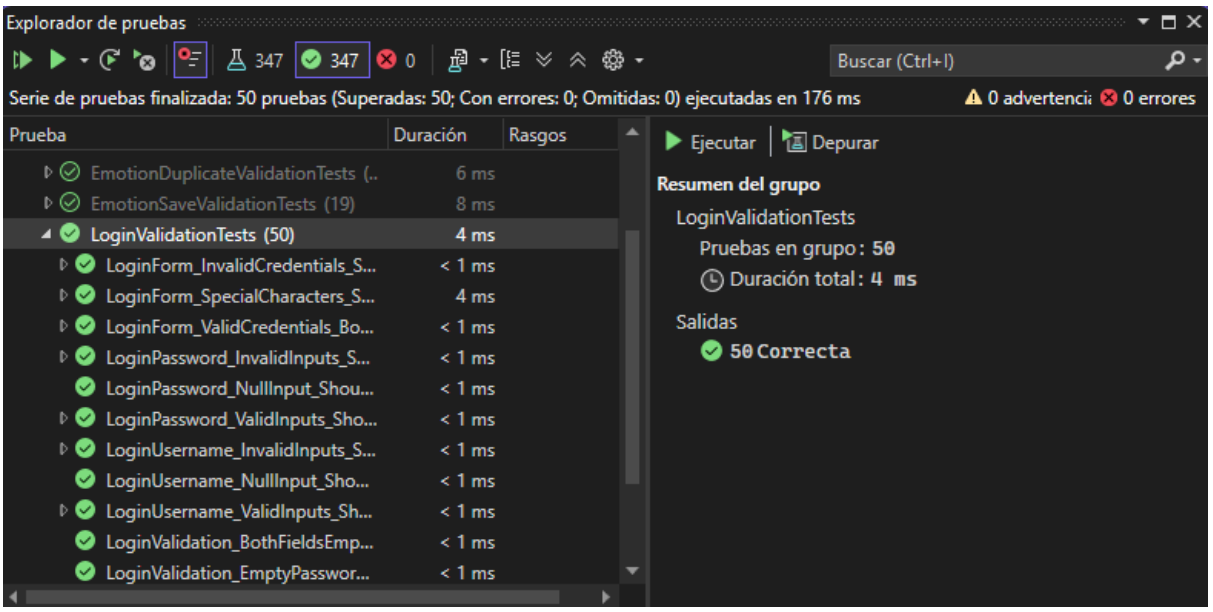
campos sean válidos simultáneamente o que se detecten errores cuando algún dato incumple las normas.




## 2. Login (LoginValidationTests.cs)

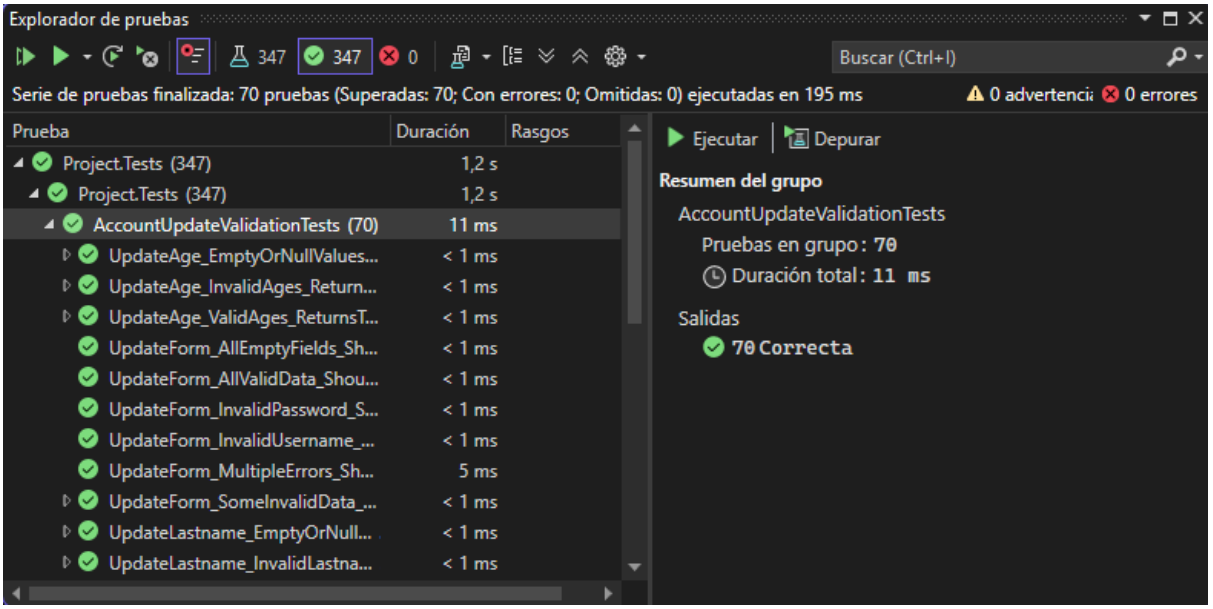
Este conjunto de pruebas unitarias valida el proceso de inicio de sesión, centrándose en la autenticación de credenciales como nombre de usuario y contraseña. Las pruebas se dividen en validaciones individuales para cada campo y escenarios completos del formulario, incluyendo casos con credenciales válidas, campos vacíos, valores nulos y caracteres especiales. Además, simula mensajes de error específicos cuando algún campo no cumple los requisitos, priorizando la detección temprana de errores (como validar primero el username).

El código también aborda casos extremos, como entradas con espacios, tabs, caracteres Unicode y combinaciones complejas.



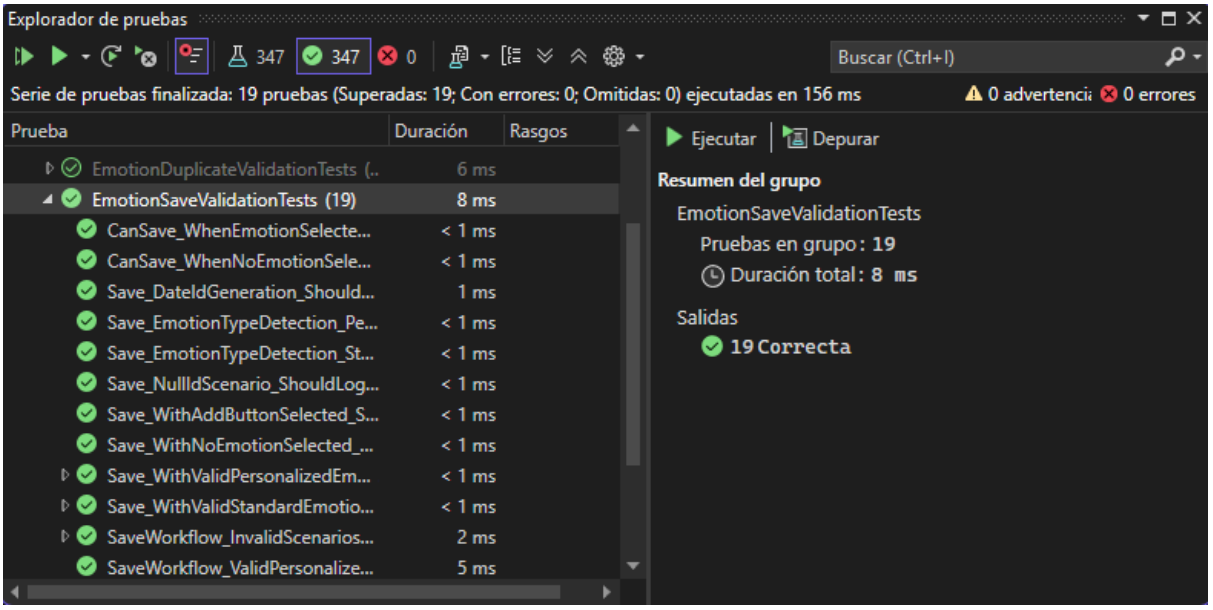

### 3. Actualización usuario (AccountUpdateValidationTests.cs) 🆕

Este código realiza pruebas unitarias para validar, de nuevo, el formulario de actualización de datos de usuario, verificando campos como nombre de usuario, contraseña, nombre, apellido y edad; comprobando que los datos cumplan reglas específicas (ej. usernames sin espacios, contraseñas seguras) y maneja casos donde los campos están vacíos o son nulos, interpretándolos como que no se desea actualizar ese dato. Las pruebas incluyen escenarios exitosos, entradas inválidas y combinaciones de errores.




### 4. Registro de emociones (EmotionSaveValidationTests.cs) 😀

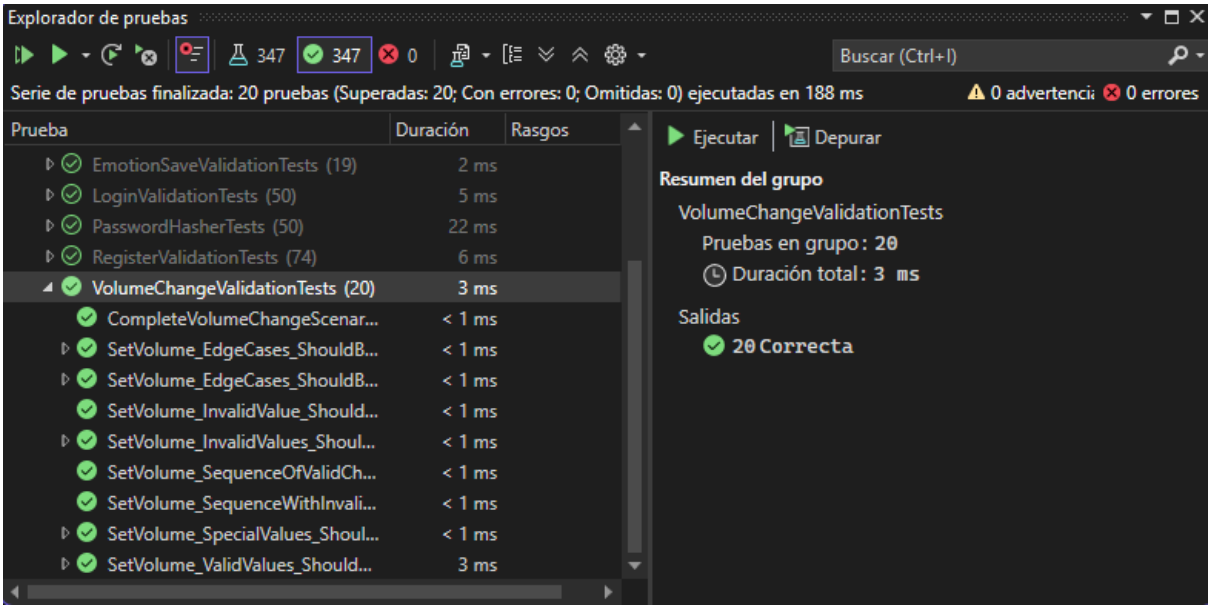
Este conjunto de pruebas valida el funcionamiento del sistema de registro de emociones, verificando tanto el guardado correcto de emociones estándar y personalizadas como el manejo de casos inválidos. Las pruebas se enfocan en tres aspectos clave: la propiedad CanSave (que debe ser true solo cuando hay una emoción válida seleccionada), el método Save (que debe procesar correctamente los diferentes tipos de emociones y rechazar casos como selección nula o del botón de agregar), y la generación automática del DateId (usando la fecha actual cuando no se proporciona una). Además, se incluyen pruebas para distinguir entre emociones estándar y personalizadas mediante la propiedad IsLocalImage.




5. **Cambio de volumen (VolumeChangeValidationTests.cs)** 🔊

Estas pruebas unitarias validan el sistema de control de volumen en un servicio de reproducción de audio, asegurando que solo se acepten valores dentro del rango permitido (0 a 1). El código prueba tres aspectos principales: valores válidos (como 0, 0.5 y 1), valores inválidos (fuera del rango, números especiales como NaN o infinito) y casos límite (valores cercanos a los bordes del rango). Cada prueba verifica si el volumen cumple con las condiciones requeridas y registra los resultados con mensajes descriptivos.

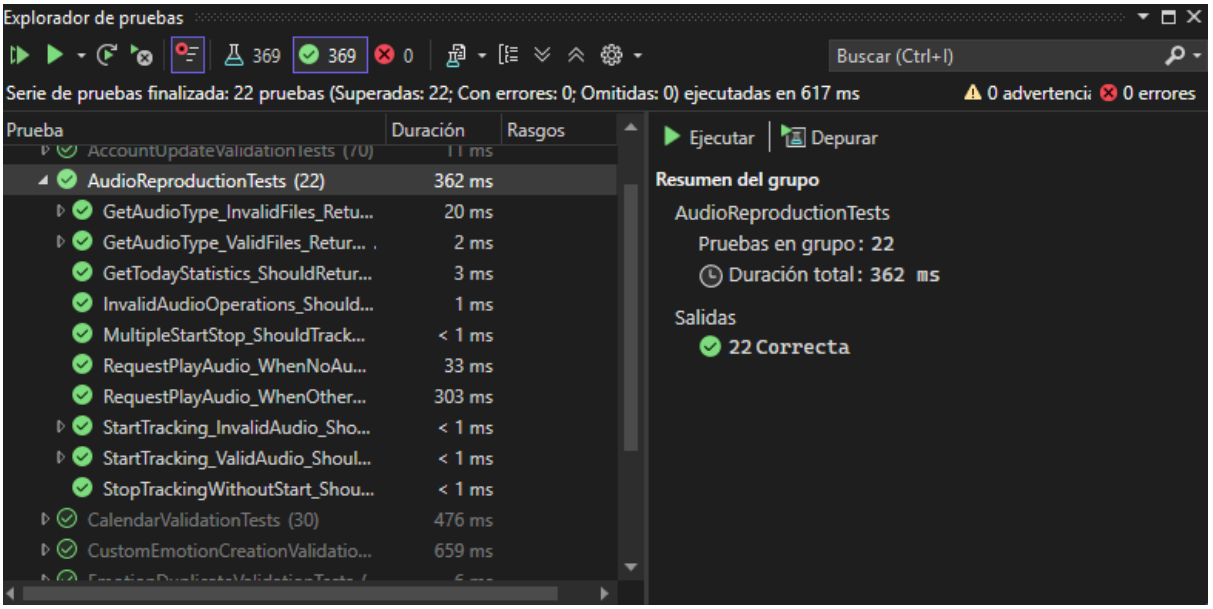
Además, el código incluye pruebas de escenarios complejos como secuencias de cambios de volumen (donde se mezclan valores válidos e inválidos) y simula mensajes de error específicos para casos no válidos.




### 6. Reproducción de audio (AudioReproductionTests.cs) 🎧

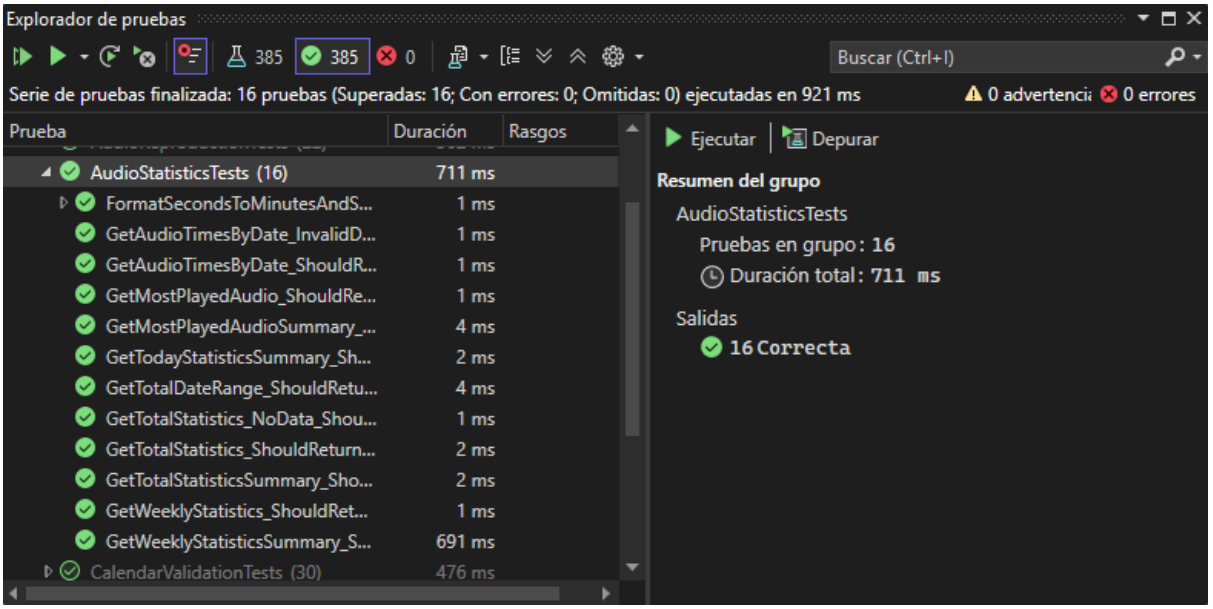
Estas pruebas unitarias verifican el sistema de reproducción y seguimiento de audio, enfocándose en tres aspectos clave: la detección del tipo de audio (como "japon", "ethno" o "piano") basada en el nombre del archivo, el manejo del estado de reproducción (inicio/detención) y el registro de estadísticas de uso. Las pruebas validan que el sistema identifique correctamente diferentes formatos de archivos de audio, gestione adecuadamente las solicitudes de reproducción (incluyendo la interrupción de audios previos) y mantenga un registro preciso del tiempo de reproducción por cada tipo de audio.

El conjunto de pruebas incluye escenarios complejos como múltiples ciclos de reproducción, operaciones inválidas (como detener un audio no iniciado) y el manejo de casos extremos (archivos con nombres desconocidos o valores nulos).



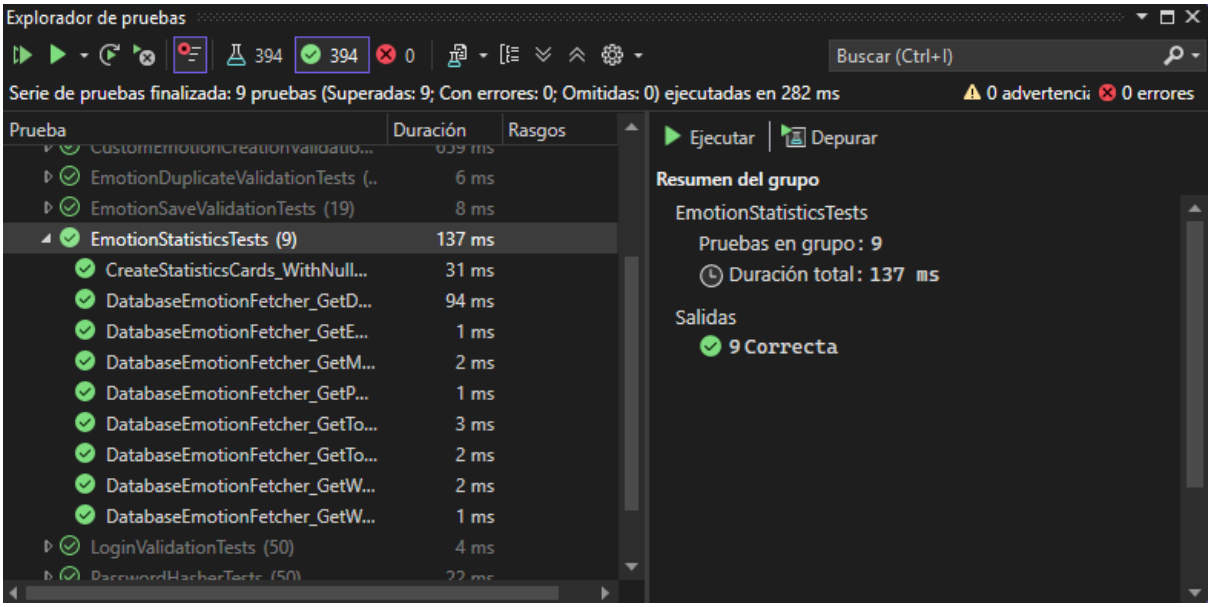

7. **Estadísticas de audio (AudioStatisticsTests.cs)** 📈

   Este conjunto de pruebas unitarias valida la generación y presentación de estadísticas de reproducción de audio, incluyendo funciones de formato (como convertir segundos a minutos y segundos), resúmenes textuales (diarios, semanales y totales) y consultas específicas por fecha. Las pruebas verifican que los métodos del DatabaseAudioFetcher devuelvan datos estructurados correctamente (tuplas con tiempos por categoría: étnico, japonés y piano) y manejen adecuadamente casos sin datos, manteniendo la coherencia en los formatos de salida y rangos de fechas.



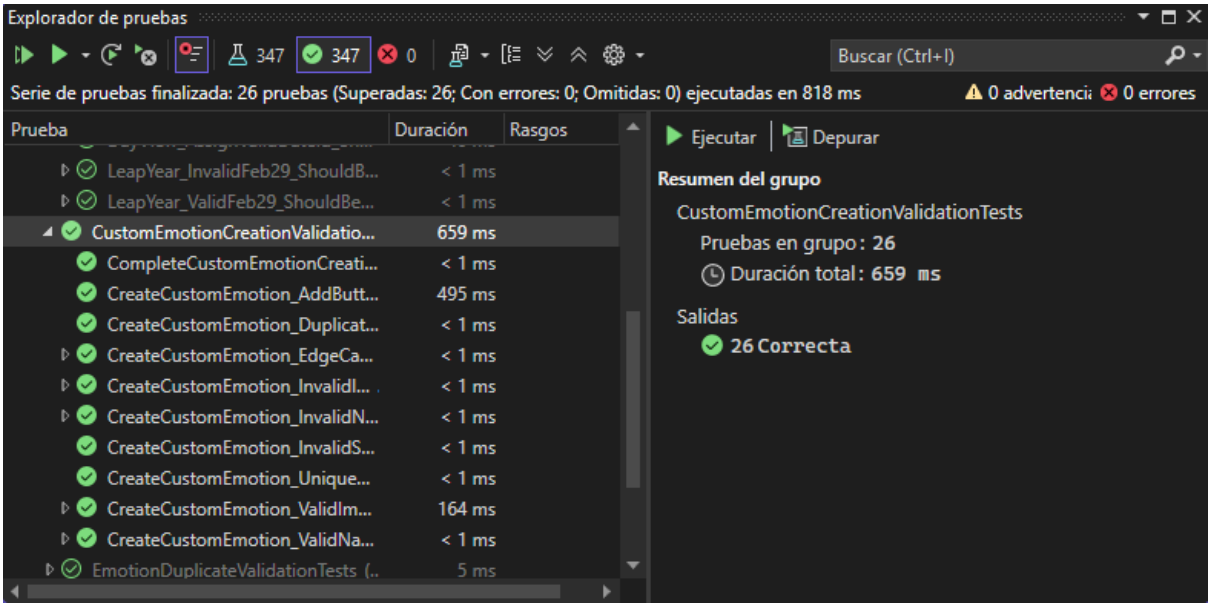

8. **Estadísticas de emociones (EmotionStatisticsTests.cs)** 📊

Este conjunto de pruebas unitarias valida la presentación y lógica de estadísticas de emociones en la aplicación, centrándose en dos aspectos principales: los componentes de interfaz de usuario y las consultas a la base de datos. Las pruebas verifican el comportamiento del componente emotionsStatistics, incluyendo su capacidad para manejar casos extremos como un contenedor nulo al crear tarjetas de estadísticas. Por otro lado, se comprueba que el DatabaseEmotionFetcher pueda recuperar correctamente diferentes conjuntos de datos (frecuencias de emociones estándar, personalizadas y semanales) y generar resúmenes textuales legibles para el usuario, manejando adecuadamente situaciones donde no hay datos disponibles.



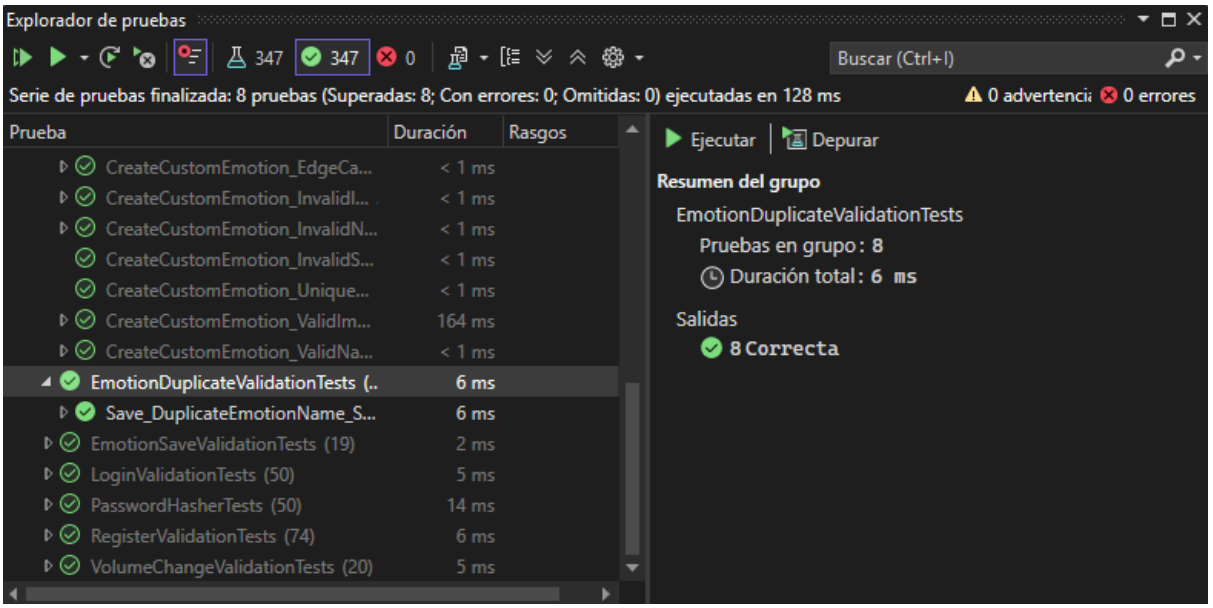

9. **Creación de emoción personalizada (CustomEmotionCreationValidationTests.cs)** ✍️

Estas pruebas unitarias validan el proceso de creación de emociones personalizadas, verificando tres aspectos clave: el nombre de la emoción (no vacío, máximo 30 caracteres y único), la ruta de la imagen (formatos .png o .jpg válidos) y la distinción del botón de agregar. Las pruebas incluyen casos válidos (nombres normales, rutas correctas), inválidos (nombres vacíos, rutas incorrectas) y casos especiales, asegurando que el sistema rechace entradas incorrectas y mantenga consistencia en los datos.




### 10. Duplicidad de emociones (EmotionDuplicateValidationTests.cs) !!

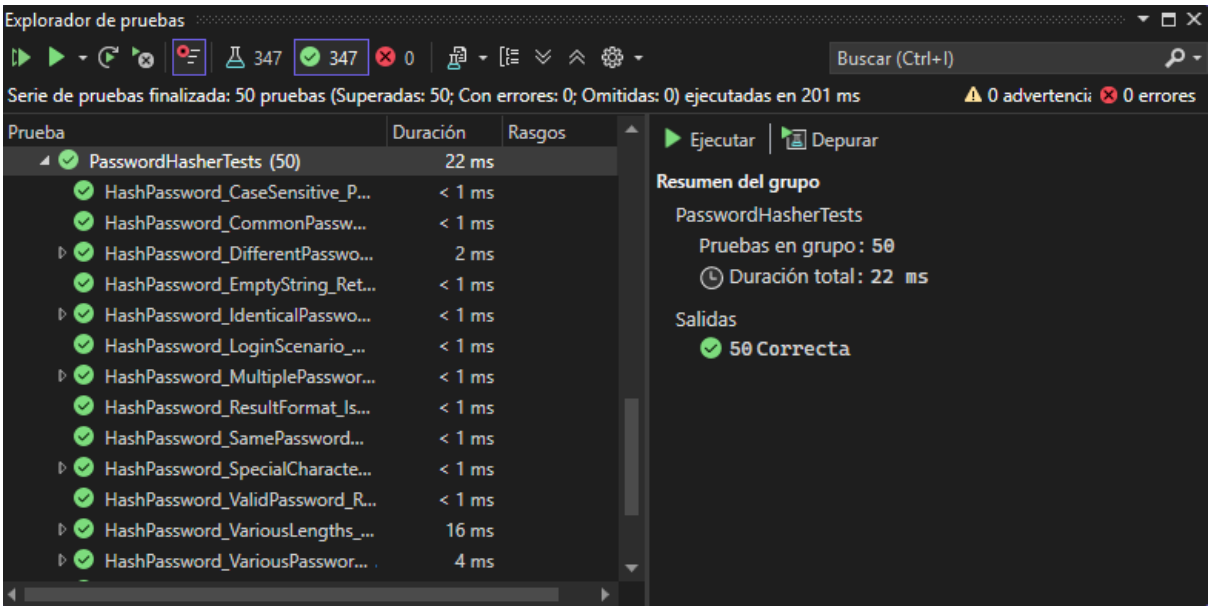
Estas pruebas unitarias verifican la detección de emociones duplicadas en el sistema, comparando nombres de emociones sin distinción entre mayúsculas y minúsculas. El código incluye una lista predefinida de emociones existentes (tanto estándar como personalizadas) y prueba diferentes variaciones de escritura para asegurar que el sistema identifique correctamente los duplicados.




### 11. Password hasher (PasswordHasherTests.cs) 🔑

Estas pruebas unitarias validan el funcionamiento de un sistema de hashing de contraseñas utilizando el algoritmo SHA-256. El código verifica que el hasher produzca resultados consistentes y seguros, incluyendo pruebas para: contraseñas

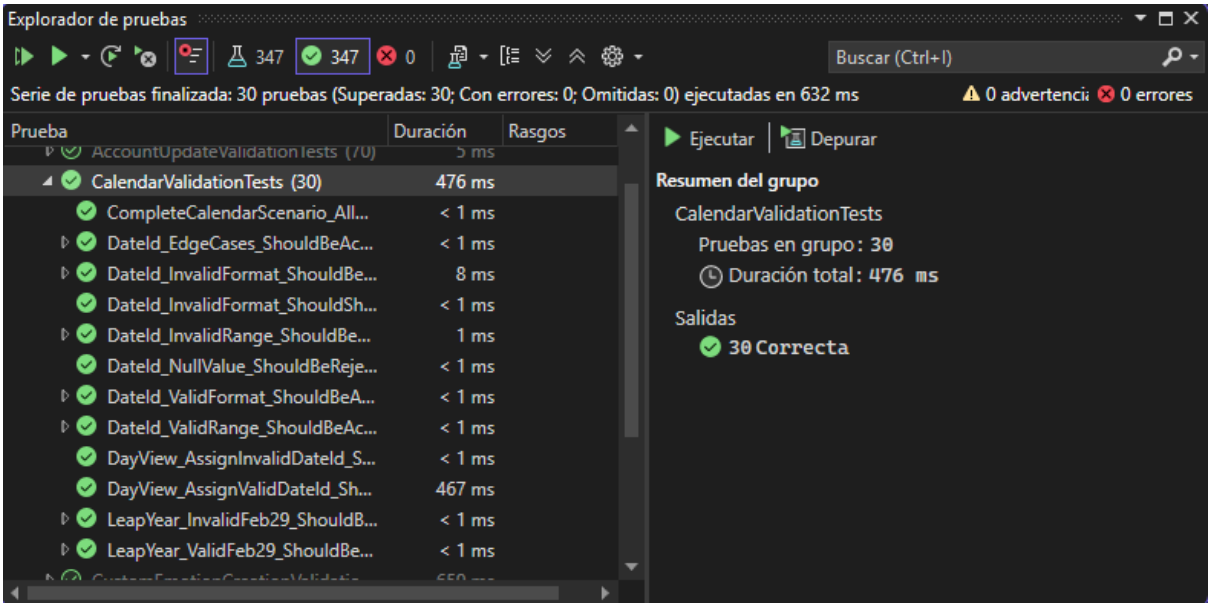
válidas (diferentes longitudes y complejidades), consistencia del hash (misma contraseña genera mismo hash), sensibilidad a mayúsculas/minúsculas, manejo de caracteres especiales, y contraseñas extremadamente largas. Cada prueba confirma que el hash resultante tenga exactamente 64 caracteres hexadecimales en minúsculas.



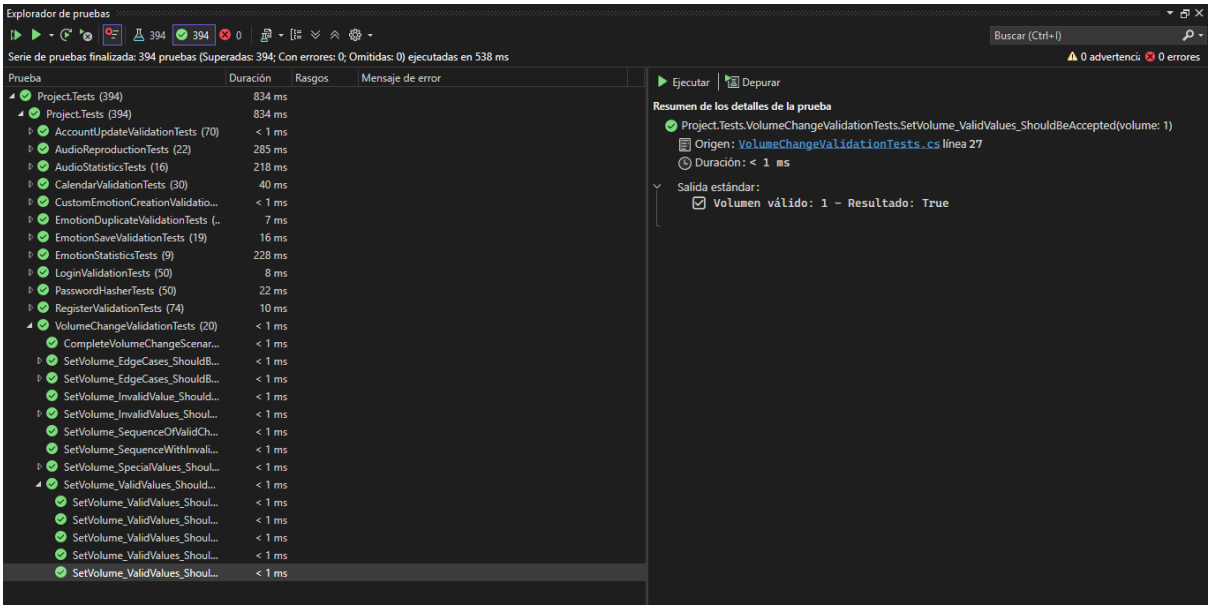

### 12. Interacción con el calendario (CalendarValidationTests.cs) 📅

Estas pruebas unitarias validan el manejo de fechas en un componente de calendario, verificando el formato correcto (yyyy-MM-dd), el rango permitido (1900-2100) y casos especiales como años bisiestos. El código prueba tanto fechas válidas (como "2024-02-29" para años bisiestos) como inválidas (formatos incorrectos, meses/días fuera de rango), asegurando que el sistema rechace adecuadamente las entradas incorrectas.

También incluye pruebas para el componente dayView, confirmando que asigne correctamente las fechas aunque sean inválidas, lo que permite un manejo flexible de errores.




Para terminar, a continuación se presenta el resumen de validez de las funcionalidades esenciales de la aplicación y la superación de todas las pruebas, dentro del entorno de Visual Studio 🖥️:



## Informe del linter ℹ️

Para el linter, utilizamos Roslyn Analyzers gracias a su enfoque en la calidad de código y estilo, de manera robusta. Especificando, se realizó lo siguiente:

- Utilizamos la configuración por defecto de Roslyn.
- El análisis encontró 94 hallazgos presentes en 39 archivos del proyecto, 73 advertencias (78%), y 21 notas informativas (21%) de los cuales las más comunes fueron los siguientes:

  -CS8618 (33 incidencias): propiedades no anulables que quedan sin inicializar al salir del constructor.

  -CS8981 (12 incidencias): nombres de tipos formados únicamente por caracteres ASCII en minúscula, lo cual choca con las convenciones de nomenclatura de C#.

  -CA1822 (10 incidencias): métodos de instancia que no usan datos de instancia y pueden convertirse en métodos estáticos.

  -CS8601 (8 incidencias): posible asignación de referencia nula a un valor que no acepta nulos.

  -CS8604 (5 incidencias): posible paso de argumento nulo a un parámetro que no acepta nulos.

- El archivo lint-report.sarif con todas las especificaciones se encuentra en Project/src/lint-report.sarif.

Teniendo este resultado en cuenta, tenemos claridad sobre las advertencias encontradas y consideramos que no requerirá mucho esfuerzo ni tiempo solucionarlas; aún así, somos conscientes de que estos “errores”, sobre todo relacionados al manejo preventivo de nulos, no debieron pasar desapercibidos para nosotros. 😔